WHAT IS CLAIMED IS:

1. A method of managing access to an array susceptible to concurrent operations on a sequence encoded therein, the method comprising:

executing as part of a pop operation, a double compare and swap (DCAS) to atomically update a then-current, end identifying index for the array and a element of the array adjacent to that identified by the end identifying index; and

returning from the DCAS, on failure thereof, an indication by which an empty state of the array is detectable.

2. The method of claim 1,

wherein the indication by which the empty state of the array is detectable is indicative of presence of a distinguishing value in the adjacent element.

3. The method of claim 1, wherein the array encodes a double-ended queue as a circular buffer of bounded size, the end identifying index and an opposing end identifying index delimiting the sequence.

4. The method of claim 1,

wherein the pop operation is a left pop operation;

wherein the end identifying index is a left-end index; and

wherein the adjacent element is to the right of the identified element.

5. The method of claim 1,

wherein the pop operation is a right pop operation;

wherein the end identifying index is a right-end index; and

wherein the adjacent element is to the left of the identified element.

6. A method of managing access to an array susceptible to concurrent operations on a sequence encoded therein, the method comprising:

executing as part of a push operation, a double compare and swap (DCAS) to atomically update a then-current, end identifying index for the array and an element of the array identified by the end identifying index; and

returning from the DCAS, on failure thereof, an indication by which a full state of the array is detectable.

7. The method of claim 6,

wherein the indication by which the full state of the array is detectable is indicative of absence of a distinguishing value in the identified element.

8. The method of claim 6,

wherein the push operation is a left push operation; and

wherein the end identifying index is a left-end index.

9. The method of claim 6,

wherein the push operation is a right push operation; and

wherein the end identifying index is a right-end index.

10. A method of providing concurrent access to a double-ended data structure of bounded size implemented using a circular buffer technique, the method comprising:

as part of an access to a first-end of the double-ended data structure, performing in alternate legs of a conditional branch:

a first multi-way compare and swap on then-current contents of a first-end index store and a corresponding element of the double-ended data structure to disambiguate a retry state and a boundary condition state of the double-ended data structure;

a second multi-way compare and swap on then-current contents of the first-end index store and a corresponding element of the double-ended data structure, the second multi-way compare and swap performing the access and, on failure thereof, returning an

indication disambiguating a retry state and the boundary condition state of the double-ended data structure,

wherein the conditional branch discriminates between presence and absence of a distinguishing value in an element of the double-ended data structure corresponding to the then-current contents of the first-end index store.

11. The method of claim 10,

wherein the access includes a pop from the first-end of the double-ended data structure;

wherein the boundary condition state is an empty state of the double-ended data structure; and

wherein the retry state results from a concurrently performed push or pop access at the first-end of the double-ended data structure.

12. The method of claim 10,

wherein the access includes a push onto the first-end of the double-ended data structure;

wherein the boundary condition state is a full state of the double-ended data structure; and

wherein the retry state results from a concurrently performed push or pop access at the first-end of the double-ended data structure.

13. The method of claim 10, wherein the double-ended data structure includes a double-ended queue (deque).

14. The method of claim 10, wherein the multi-way compare and swap is a double compare and swap (DCAS).

15. A method of managing concurrent access to a double-ended queue (deque), the method comprising:

employing, in an implementation of a pop operation, execution of a double compare and swap (DCAS) to interrogate instantaneous values of a first end index and a deque element adjacent to that identified thereby

for a signature indicative of an empty state of the array, the signature including presence in that adjacent element of a distinguishing value,

wherein successful execution of an opposing end pop operation includes execution of a DCAS to atomically update a second end index and a deque element adjacent to that identified thereby, the update of that adjacent element storing the distinguishing value therein.

16. The method of claim 15, further comprising:

wherein successful execution of a competing, same end pop operation includes execution of a DCAS to atomically update the first end index and a deque element adjacent to that identified thereby, the update of that adjacent element storing the distinguishing value therein.

17. The method of claim 15, further comprising:

wherein the first end index is a left index and, if the state of the deque is non-empty, the deque element adjacent to that identified thereby is a left most element of the deque;

wherein the second end index is a right index and, if the state of the deque is non-empty, the deque element adjacent to that identified thereby is the right most element of the deque.

18. The method of claim 15,

wherein the pop operation is a left pop operation and the opposing end pop operation is a right pop operation; and

wherein the first end index is a left end index and the element adjacent to that identified thereby is adjacent to the right.

19. The method of claim 15, wherein the distinguishing value is encoded as a null value.

20. The method of claim 15, further comprising:

employing, in an implementation of a push operation, execution of a double compare and swap (DCAS) to interrogate instantaneous values of a third end index and a deque element identified thereby for a signature

indicative of an full state of the deque, the signature including absence in that identified deque element of a distinguishing value,

wherein successful execution of an opposing end push operation includes execution of a DCAS to atomically update a fourth end index and a deque element identified thereby, the update of the identified deque element storing a value other than the distinguishing value therein.

21. The method of claim 20,

wherein the first end index and the third end index identify a same end of the deque; and

wherein the second end index and the fourth end index identify a same end of the deque.

22. The method of claim 20,

wherein the first end index and the fourth end index identify a same end of the deque; and

wherein the second end index and the third end index identify a same end of the deque.

23. A method of managing concurrent access to a double-ended queue (deque), the method comprising:

employing, in an implementation of a push operation, execution of a double compare and swap (DCAS) to interrogate instantaneous values of a first end index and a deque element identified thereby for a signature indicative of a full state of the deque, the signature including absence in that identified deque element of a distinguishing value,

wherein successful execution of an opposing end push operation includes execution of a DCAS to atomically update an opposing end index and a deque element identified thereby, the update of the identified deque element storing a value other than the distinguishing value therein.

24. The method of claim 23, further comprising:

Cont A1

wherein successful execution of a competing, same end push operation includes execution of a DCAS to atomically update the first end index and a deque element identified thereby, the update of that adjacent element storing a value other than the distinguishing value therein.

25. A method of managing concurrent access to an array susceptible to competing accesses at same and opposing ends thereof, the method comprising:

executing as part of a first access operation, a double compare and swap (DCAS) to atomically update a first end identifying index and an element of the array corresponding to a then-current value thereof;

executing as part of a competing second access operation, a DCAS to atomically update a second end identifying index and an element of the array corresponding to a then-current value thereof,

wherein, if successful completion of one of the first and the second competing access operations results in a boundary condition state of the array, the DCAS of the other of the first and the second access operations fails and returns an indication thereof.

26. The method of claim 25,

wherein the first access operation and the competing second access operation are competing pop operations;

wherein the array elements corresponding to the first and second indices are each adjacent to that identified by the respective index;

wherein the boundary condition state is an empty state; and

wherein the adjacent element referenced by the failing one of the competing pop operations encodes a distinguishing value signifying the empty state.

27. The method of claim 25,

wherein the competing pop operations are competing opposing end pop operations; and

wherein the first index and the second index identify opposing ends of the array;

Count A1

28. The method of claim 25,
wherein the competing pop operations are competing same end pop operations; and
wherein the first index and the second index identify a same end of the array;

29. The method of claim 25,
wherein the first access operation and the competing second access operation are competing push operations;
wherein the array elements corresponding to the first and second indices are each identified by the respective index;
wherein the boundary condition state is an full state; and
wherein the array element referenced by the failing one of the competing push operations encodes a value other than a distinguishing value.

30. The method of claim 25,
wherein the competing push operations are competing opposing end push operations; and
wherein the first index and the second index identify opposing ends of the array;

31. The method of claim 25,
wherein the competing push operations are competing same end push operations; and
wherein the first index and the second index identify a same end of the array;

32. A double-ended queue (deque) implementation comprising:
a contiguous array S of bounded size encoded in an addressable store;
a left index L and a right index R into the contiguous array, the contiguous array S, the left index L and the right index R together defining a circular buffer with state including a sequence of zero or more values encoded in the contiguous array between elements S[L] and S[R] thereof; and

Cont A1

a computer readable encoding of at least a first access operation, execution of the first access operation operating at a particular end of the sequence and employing a double compare and swap (DCAS) to atomically update a corresponding one, but not both, of the left and right indices L and R and an element of the contiguous array adjacent to the contiguous array element identified thereby.

33. The double-ended queue (deque) implementation of claim 32,
wherein the first access operation includes a push; and
wherein, on failure, the DCAS returns an indication by which a full state of the contiguous array is detected.

34. The double-ended queue (deque) implementation of claim 32,
wherein the first access operation includes a pop; and
wherein, on failure, the DCAS returns an indication by which an empty state of the contiguous array is detected.

35. The double-ended queue (deque) implementation of claim 32, further comprising:
computer readable encodings of at least three additional access operations,
wherein the first and the three additional access operations together include push and pop operations at left and rights end of the sequence, respectively.

36. A concurrent shared object implementation comprising:
a contiguous array encoded in an addressable store;
opposing indices into the contiguous array usable to delimit therebetween a portion of the contiguous array for storage of a sequence of zero or more data values; and
a computer readable encoding of push and pop operations defined to operate on elements of the contiguous array and on respective of the opposing indices,

Cont A1

wherein the push operation employs a first instance of a double compare and swap (DCAS) operation to atomically update one of the opposing indices and a corresponding element of the contiguous array while returning on failure, an indication by which a full state of the contiguous array is detected, and

wherein the pop operation employs a second instance of a DCAS operation to atomically update one of the opposing indices and a corresponding element of the contiguous array while returning on failure, an indication by which an empty state of the contiguous array is detected.

37. The concurrent shared object implementation of claim 36,
wherein concurrent shared object includes a deque; and
wherein the computer readable encoding of push and pop operations includes:
opposing end variants of the push operation; and
opposing end variants of the push operation.

38. The concurrent shared object implementation of claim 36,
wherein concurrent shared object includes a queue or FIFO; and
wherein the computer readable encoding of push and pop operations operate on opposing ends of the queue or FIFO.

39. The concurrent shared object implementation of claim 36,
wherein concurrent shared object includes a stack or LIFO; and
wherein the computer readable encoding of push and pop operations operate on a same end of the stack or LIFO.

40 A computer program product encoded in at least one computer readable medium, the computer program product comprising:
at least one functional sequence implementing an access operation on a concurrent shared object, the concurrent shared object instantiable circular buffer of bounded size implementing a contiguous array delimited by a pair of end identifying indices;

Cont A1

instances of the at least one functional sequence concurrently executable by plural processors of a multiprocessor and each including a double compare and swap (DCAS) to atomically update a corresponding one of the end identifying indices and an element of the array corresponding to a then-current value thereof; and

the DCAS of the at least one functional sequence responsive to a corresponding boundary condition state of the concurrent shared object.

41. A computer program product as recited in 40,

wherein the at least one functional sequence includes opposing end variants of push and pop operations on the concurrent shared object;

wherein the boundary condition state corresponding to push operations is a full state of the array; and

wherein the boundary condition state corresponding to pop operations is an empty state of the array.

42. A computer program product as recited in 40,

wherein the at least one computer readable medium is selected from the set of a disk, tape or other magnetic, optical, or electronic storage medium and a network, wireline, wireless or other communications medium.

43. An apparatus comprising:

plural processors;

a store addressable by each of the plural processors;

first- and second-end index stores accessible to each of the plural processors for identifying opposing ends of a bounded-size contiguous array encoded in circular buffer form in the addressable store; and

means for coordinating competing access operations, the coordinating means employing in each instance thereof, at least one double compare and swap (DCAS) operation to disambiguate a retry state and a boundary condition state of the array based on then-current contents of one, but

Cont A1

not both, of first- and second-end index stores and an array element corresponding thereto.